Compiling planning into quantum optimization problems: a comparative study

Bryan O'Gorman, Eleanor G. Rieffel, Minh Do, Davide Venturelli, Jeremy Frank
NASA Ames Research Center
Moffett Field, CA 94035
firstname.lastname@nasa.gov

Abstract

One approach to solving planning problems is to compile them to another problem for which powerful off-the-shelf solvers are available; common targets include SAT, CSP, and MILP. Recently, a novel optimization technique has become available: quantum annealing (QA). QA takes as input problem instances encoded as Quadratic Unconstrained Binary Optimization (QUBO). Early quantum annealers are now available, and more sophisticated quantum annealers will likely be built over the next decades. Specific quantum annealing hardware implementations have specific constraints, restricting the types of QUBOs each can take as input. In this paper, we introduce the planning community to the key steps involved in compiling planning problems to quantum annealing hardware: a hardware-independent step, mapping, and a hardware-dependent step, embedding. After describing two approaches to mapping general planning problems to QUBO, we provide preliminary results from running an early quantum annealer on a parameterized family of hard planning problems. The results show that different mappings can lead to a substantial difference in performance, even when many superficial features of the resulting instances are similar. We also provide some insights gained from this early study, and suggest directions for future work.

Introduction

One approach to solving planning problems is to compile them to another problem for which powerful off-the-shelf solvers are available; common targets include SAT, CSP, and MILP. Recently, a novel optimization technique has become available: quantum annealing. Quantum annealing is one of the most accessible quantum algorithms for a computer science audience not versed in quantum computing because of its close ties to classical optimization algorithms such as simulated annealing.

While large-scale universal quantum computers are likely decades away from realization, we expect to see a variety of special-purpose quantum-computational hardware emerge within the next few years. Already, early quantum annealers are available, and more sophisticated quantum annealers will be built over the next decades. While certain classes of problems are known to be more efficiently solvable on a universal quantum computer (Rieffel and Polak 2011; Nielsen and Chuang 2001), for the vast majority of problems the computational power of quantum computing is unknown. Until quantum hardware became available it was impossible to empirically evaluate heuristic quantum algorithms such as quantum annealing.

While there are intuitive reasons why quantum annealing may be able to outperform classical methods on some classes of optimization problems, the effectiveness of quantum annealing is as yet poorly understood. Our work is the first to explore the use of quantum annealing to attack problems arising in planning and scheduling. This work explores compilation of planning problems to quadratic unconstrained binary optimization (QUBO) problems, the type of problem that quantum annealers are designed for. While the immaturity of the technology means that current results are limited, the significant performance differences that result from different compilation approaches suggest that subtle issues are at play in determining the best compilation approaches for quantum annealers.

In this paper, we introduce the planning community to the key steps involved in compiling planning problems to quantum annealing hardware. Figure 1 shows the main steps in our framework of solving STRIPS planning problems (Fikes and Nilsson 1972; Ghallab, Nau, and Traverso 2004) represented in PDDL using a D-Wave quantum annealer housed at NASA Ames Research Center: **mapping** the problems to QUBO, and **embedding**, which takes these hardware-independent QUBOs to other QUBOs that matches the specific quantum annealing hardware that will be used. While debate continues as to the extent to which the D-Wave machine is quantum (Johnson et al. 2011b; Boixo et al. 2013; Smolin and Smith 2013; Wang et al. 2013; Boixo et al. 2014; Shin et al. 2014b; Vinci et al. 2014; Shin et al. 2014a), these machines provide the first opportunity for researchers to experiment with quantum annealing. This work does not aim to contribute to that debate, but rather examines different mappings of application problems to quantum annealing to give insight into their relative strengths and weaknesses as best we can with current technology.

We describe two approaches to mapping general STRIPS planning problems to QUBO problems. The mappings were described in (Rieffel et al. 2014b), where one is a variant of the mapping described in (Smelyanskiy et al. 2012). We explore the properties of these mappings for a parametrized family of scheduling-type planning problems based on graph coloring (Rieffel et al. 2014a). We discuss preliminary re-

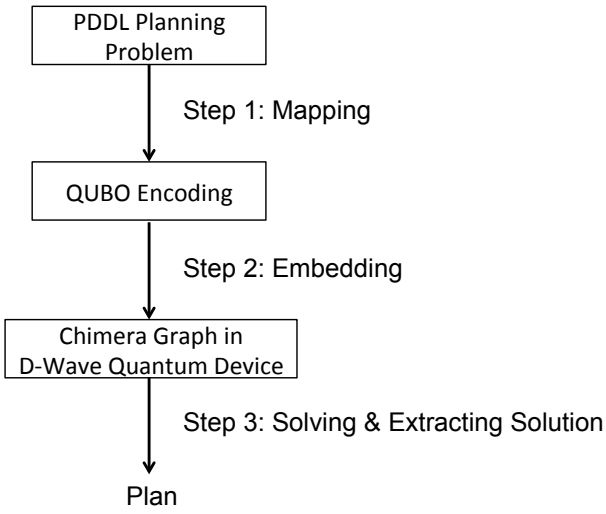

Figure 1: The main steps in our approach of using quantum annealer to solve a planning problem.

sults from an early quantum annealer, a D-Wave Two machine, applied to these problems under the two general mappings. Due to the current hardware limitation of existing quantum annealers, our empirical evaluation has been conducted on a limited set of small planning problems. Nevertheless, these early results show that different compilations to QUBO can lead to substantial differences in performance, even when many features of both the mapping and embedded QUBOs are similar.

Our paper is a reworking and deepening of our paper (Rieffel et al. 2014b) to target planning and scheduling researchers, rather than quantum computing researchers. Our main contributions are:

- A description of quantum annealing with example applications to planning;
- Two different ways of compiling general planning problems to QUBO; and
- Results from runs of a parametrized family of scheduling-type planning problems on an early quantum annealer.

We begin with an overview of quantum annealing, including the mapping and embedding compilation steps.

Ingredients of Quantum Annealing

Quantum annealing (Farhi et al. 2000; Das and Chakrabarti 2008; Johnson et al. 2011a; Smelyanskiy et al. 2012) is a metaheuristic for solving optimization problems which bears some resemblance to simulated annealing, a classical metaheuristic. Quantum annealers are special-purpose devices designed to run only this type of quantum algorithm. Other types of quantum algorithms are known that take on quite a different form, and are aimed at solving other types of problems. Quantum annealing can be applied to any optimization problem that can be expressed as a quadratic unconstrained binary optimization (QUBO) problem (Choi 2008;

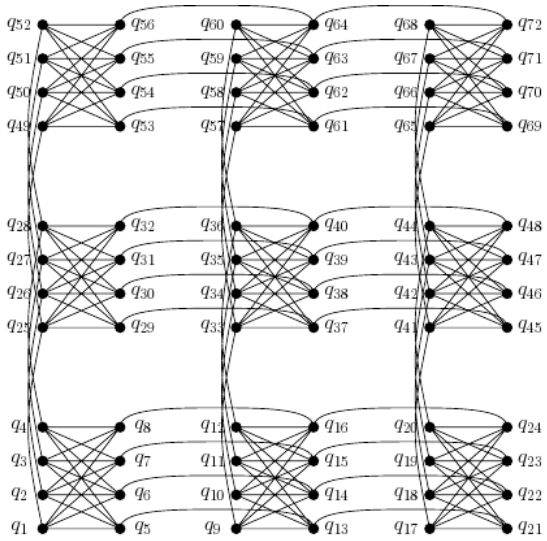

Figure 2: **The $(3,4)$-Chimera graph.** A schematic diagram from (Smelyanskiy et al. 2012) of the (M,L)-Chimera graph underlying D-Wave's architecture. In the $(3,4)$-Chimera graph shown, there are $M^2 = 9$ unit cells, each of which is a fully-connected bipartite graph $K_{4,4}$ containing $2L$ qubits. The qubits in the left column of each unit cell are connected to the analogous qubits in the unit cells above and below and the qubits in the right column of each unit cell are connected to the analogous qubits in the unit cells to the right and left.

Smelyanskiy et al. 2012; Lucas 2013). Quantum annealing is motivated by the possibility that quantum effects such as tunneling allow for efficient exploration of the cost-function landscape in ways unavailable to classical methods.

Input for quantum annealing: QUBO problems

QUBO problems are minimization problems with cost functions of the form

$$q(z_1,\ldots,z_N) = -\sum_{i=1}^{N} h_i z_i + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} J_{i,j} z_i z_j, \quad (1)$$

where the z_i are binary variables. A QUBO can be easily translated to an Ising Hamiltonian, the form of input a quantum annealer takes, through the linear transformation: $z_i = \frac{1}{2}(s_i + 1)$.

The simplicity of the QUBO formalism belies its expressivity. There exist many techniques for mapping more complicated problems to QUBO:

- Many optimization problems can be expressed in terms of cost functions that are polynomials over finite sets of binary variables. Any such function can be re-expressed, through degree-reduction techniques using ancilla variables, as quadratic functions over binary variables. We describe such degree-reduction technique in our section on the CNF mapping of planning problems to QUBO below.

- Cost functions involving non-binary, but finite-valued, variables can be rewritten in terms of binary variables alone, and optimization problems with constraints can often be written entirely in terms of cost functions over binary variables through the introduction of slack variables.

For these reasons, the QUBO setting is more general than it may, at first, seem.

Before describing the more complex QUBO mappings for general STRIPS planning problems, we give an example of a simple mapping from graph coloring to QUBO to give a feel of how mapping to QUBO works.

Example 1: Mapping of Graph Coloring, in which all vertices are to be colored so any two vertices connected by an edge have different colors, to QUBO.

Let $G = (V, E)$ be a graph with $n = |V|$ vertices, where E is the set of edges. The QUBO problem corresponding to the graph coloring problem with k colors on graph G, will have kn binary variables, z_{ic}, where $z_{ic} = 1$ means that vertex i is colored with color c, and $z_{ic} = 0$ means it is not.

The QUBO contains two different types of penalty terms. The first corresponds to the constraint that each vertex must be colored by exactly one color: $\sum_{c=1}^{k} z_{ic} = 1$. So for each vertex i, we have a term

$$\left(1 - \sum_{c=1}^{k} z_{ic}\right)^2.$$

The second corresponds to the constraint that two vertices connected by an edge cannot be colored with the same color. For each vertex i, we have a term

$$\sum_{(i,j)\in E} \sum_{c=1}^{k} z_{ic} z_{jc}.$$

Altogether, the QUBO is

$$\sum_{i=1}^{n} \left(1 - \sum_{c=1}^{k} z_{ic}\right)^2 + \sum_{(i,j)\in E} \sum_{c=1}^{k} z_{ic} z_{jc}.$$

Example 2: Mapping of Hamiltonian Path problem, in which the goal is a path that visits each vertex in a graph $G = (V, E)$ exactly once, to QUBO.

For a Hamiltonian path problem with n sites, we have n^2 variables

$$\{x_{11}, \ldots, x_{1n}, x_{21}, \ldots, x_{2n}, \ldots, x_{n1}, \ldots, x_{nn}.\}$$

The subscripted indices indicate, in order, a site and the time slot in which it is visited; $x_{ij} = 1$ means that the ith site is the jth site visited, and $x_{ij} = 0$ means that the ith site is not visited in the jth time slot.

There are three types of terms in the QUBO cost function. The first type of term enforces that each site is visited exactly once. Thus, for each site i, we will have a constraint:

$$\left(\sum_{j=1}^{n} x_{ij} - 1\right)^2.$$

The second type of term enforces that in each time slot no more than one site is visited. Thus, for each time slot j:

$$\left(\sum_{i=1}^{n} x_{ij} - 1\right)^2.$$

The third type of term is a single term penalizing the violation of edge constraints. It penalizes visiting the i'th site right after the ith site if they are not connected by an edge:

$$\sum_{j=1}^{j=n-1} \sum_{\{i,i'|(i,i')\notin E\}} x_{ij} x_{i',j+1}.$$

Embedding QUBOs in Specific Quantum Annealing Hardware

As outlined for planning in Figure 1, once an application instance has been mapped to a QUBO problem, a second step is required to compile it to the specific quantum annealing hardware that will be used. Typically, each quantum device has a set of physical quantum bits (qubits) that are linked in a certain way. Ideally, each binary variable z_i in a QUBO formula would be represented by a single qubit q_i of the machine. Hardware constraints place limits, however, on which qubits can be connected to which other qubits.

The strength of the coupling between two qubits q_i and q_j representing two binary variables z_i and z_j can model the term $J_{i,j} z_i z_j$ in the QUBO formula 1 introduced above. The D-Wave processors use a Chimera architecture in which each qubit is connected to at most 6 other qubits (Fig. 2), so any QUBO variable that appears in more than 6 terms must be represented by multiple physical qubits in order for the problem to be implemented in this architecture. The D-Wave Two used in the experiments has a $(8, 4)$-Chimera graph architecture, but with 3 broken qubits that are not used. Other limitations, beyond the degree 6 constraint, exist as well. Therefore, each logical qubit must be mapped to a connected set of physical qubits, which, together with the connecting edges, is called the logical qubit's *vertex-model*. The overall mapping of logical qubits and couplers to physical ones is called a *model*, as discussed below.

Consider, for example, a simple QUBO

$$z_1 z_2 + z_1 z_3 + z_2 z_3,$$

which can be represented as a triangle, with the three variables as the vertices, and the edge between each pair of vertices indicating a quadratic term of the QUBO. Ideally, we would represent each of these variables by qubits q_1, q_2, and q_3 with hardware connections between each pair so that the three terms in the QUBO can be directly realized in the hardware. Fig. 2 shows the qubit connections for the type of quantum annealing architecture we used in these experiments. In that graph, no three qubits are all mutually connected to each other. The best we can do is to use four qubits to represent the three variables z_1, z_2, and z_3. We may take, for example, z_1 to be represented by q_{52} and q_{56}, z_2 to be represented by q_{51}, and z_3 to be represented by q_{55}, so that there is a connection corresponding to each of the three

terms in the QUBO: the term $z_1 z_2$ can be implemented using the connection between qubits q_{56} and q_{51}, the term $z_1 z_3$ can be implemented using the connection between qubits q_{52} and q_{55}, and the term $z_1 z_2$ can be implemented using the connection between qubits q_{51} and q_{55}. We will also need to use the connection between qubits q_{52} and q_{56} to enforce that the two qubits take on the same value since together they are meant to represent a single variable.

Mapping General STRIPS Planning Problems to QUBO

In this section, we describe two different mappings from a general class of planning problems to QUBO. Specifically, we consider STRIPS planning problems, classical planning problems that are expressed in terms of binary state variables and actions. The first mapping takes a time-slice approach. The second approach first maps a planning problem to SAT, and then reduces higher order terms to quadratic terms through a series of gadgets. Our mappings allow both positive and negative preconditions.

Time-slice Mapping

This mapping from general classical planning problems to QUBO form is a variant of the one developed and described in (Smelyanskiy et al. 2012). This approach shares many similarities with existing compilation approaches (to SAT, CSP, MILP etc.) derived from the Plan Graph (Blum and Furst 1997). Specifically, it presets a horizon L and then encode the interleaving proposition and action layers up to the preset level L.

If the original planning problem has N state variables x_i and M actions y_j and we are looking for a plan of length L, then we define a time-slice QUBO problem in terms of $N(L+1)+LM$ binary variables. There are two groups of binary variables. The first group consists of $N(L+1)$ binary variables $x_i^{(t)}$ that indicate whether the state variable x_i is 0 or 1 at time step t, for $t \in \{0, \ldots, L\}$. The second group consists of LM binary variables $y_j^{(t)}$ that indicate whether or not the action y_j is carried out between time steps $t-1$ and t.

The total cost function is written as a sum

$$\begin{aligned} H &= H_{\text{initial}} + H_{\text{goal}} + H_{\text{precond}} \\ &\quad + H_{\text{effects}} + H_{\text{no-op}} + H_{\text{conflicts}}. \end{aligned}$$

The first two terms capture the initial condition and the goal condition. Let $\mathcal{I}^{(+)}$ be the set of state variables that are 1 in the initial condition and $\mathcal{I}^{(-)}$ be the set of state variables that are initially set to 0. Similarly, let $\mathcal{G}^{(+)}$ (resp. $\mathcal{G}^{(-)}$) be the set of goal variables with value 1 (resp. 0). To capture the requirement that a plan start in the appropriate initial state and meets the goals, we include in the cost function:

$$H_{\text{initial}} = \sum_{i \in \mathcal{I}^{(+)}} \left(1 - x_i^{(0)}\right) + \sum_{i \in \mathcal{I}^{(-)}} x_i^{(0)}$$

and

$$H_{\text{goal}} = \sum_{i \in \mathcal{G}^{(+)}} \left(1 - x_i^{(L)}\right) + \sum_{i \in \mathcal{G}^{(-)}} x_i^{(L)}.$$

We next add terms to the cost function that penalize a plan if an action is placed at time t but the prior state does not have the appropriate preconditions:

$$\begin{aligned} H_{\text{precond}} &= \sum_{t=1}^{L} \sum_{j=1}^{M} \left(\sum_{i \in \mathcal{C}_j^{(+)}} \left(1 - x_i^{(t-1)}\right) y_j^{(t)} \right. \\ &\quad \left. + \sum_{i \in C_j^{(-)}} x_i^{(t-1)} y_j^{(t)} \right), \end{aligned}$$

where $\mathcal{C}_j^{(+)}$ is the set of positive preconditions for action j and $\mathcal{C}_j^{(-)}$ is the set of negative preconditions.

Next, we must penalize variable changes that are not the result of an action. We start with this term, the $H_{\text{no-op}}$ term, that penalizes variable changes:

$$H_{\text{no-op}} = \sum_{t=1}^{L} \sum_{i=1}^{N} \left[x_i^{(t-1)} + x_i^{(t)} - 2 x_i^{(t-1)} x_i^{(t)} \right].$$

This term gives a cost penalty of 1 every time a variable is flipped. Of course, when the effect of an action does result in a variable flipping, we do not want this penalty, so we will make up for this penalty when we add the term that corresponds to the effects of an action. Specifically, we need to penalize if the subsequent state does not reflect the effects of a given action. Let $\mathcal{E}_j^{(+)}$ be the set of positive effects for action j and $\mathcal{E}_j^{(-)}$ the set of negative effects. The penalty if the appropriate effects do not follow the actions is captured by the following term:

$$\begin{aligned} H_{\text{effects}} &= \sum_{t=1}^{L} \sum_{j=1}^{M} \left(\sum_{i \in \mathcal{E}_j^{(+)}} y_j^{(t)} \left(1 + x_i^{(t-1)} - 2 x_i^{(t)}\right) \right. \\ &\quad \left. + \sum_{i \in \mathcal{E}_j^{(-)}} y_j^{(t)} \left(2 x_i^{(t)} - x_i^{(t-1)}\right) \right). \end{aligned}$$

In order to understand this term, we must consider it together with the no-op term. When $y_j^{(t)} = 1$, the corresponding term for $i \in \mathcal{E}_j^{(+)}$ (resp. $i \in \mathcal{E}_j^{(-)}$), taken together with the no-op term, can be written

$$\left(1 + 2 x_i^{(t-1)}\right) \left(1 - x_i^{(t)}\right)$$

(resp.

$$\left(3 - 2 x_i^{(t-1)}\right) x_i^{(t)}$$

for negative effects), resulting in a positive penalty unless $x_i^{(t)} = 1$ (resp. $x_i^{(t)} = 0$). By using this form we have corrected for the corresponding no-op term.

Parallel Plans: Classical planners often allow for parallel plans in which more than one action can take place at one

time if those actions could have been done in any order. Encodings that allow parallel plans are often significantly smaller due to the big reduction in the preset horizon value L. The QUBO encoding described so far works fine for domain with linear plans, but when more than one action can take place at a given time, we are in danger of overcorrecting for the no-op term. If multiple actions at the same time have the same effect, the H_{effects} term will add a term for each of those actions, thus overcompensating for the no-op penalty. To avoid overcompensating, we penalize multiple actions at the same time having the same effect, discouraging all such actions[1]. To ensure that two actions that conflict in the sense that positive preconditions of one overlap with negative effects of the other or vice versa, and to avoid overcompensating, we include the penalty

$$H_{\text{conflict}} = \sum_{t=1}^{L}\sum_{i=1}^{N}\left(\sum_{\left\{j\middle|i\in\mathcal{C}_j^{(+)}\cup\mathcal{E}_j^{(-)}\right\}}\sum_{\left\{j'\neq j\middle|i\in\mathcal{E}_{j'}^{(-)}\right\}} y_j^{(t)}y_{j'}^{(t)} + \sum_{\left\{j\middle|i\in\mathcal{C}_j^{(-)}\cup\mathcal{E}_j^{(+)}\right\}}\sum_{\left\{j'\neq j\middle|i\in\mathcal{E}_{j'}^{(+)}\right\}} y_j^{(t)}y_{j'}^{(t)}\right).$$

Encoding Size Improvements: While for explanatory purposes it was useful to include variables for the state at time $t = 0$, those specified by initial conditions can be set ahead of time, so that we don't need to include the H_{initial} term. The same is true of the H_{goal} term. We can also replace all of their occurrences in $H_{\text{no-op}}$, H_{precond}, and H_{effect} with these set values to simplify those constraints. Furthermore, reachability and relevant analysis starting from the initial and goal states, preprocessing techniques employed by compilation-based planners such as Blackbox (Kautz and Selman 1999) and GP-CSP (Do and Kambhampati 2001), can be used to remove or preset the values of variables in different layers. These simplifications result in modified terms $H'_{\text{no-op}}$, H'_{precond}, and H'_{effects}. Additionally, since in our setting we have followed the convention that preconditions must be positive, we can use a simpler version of the H_{precond} term:

$$H'_{\text{precond}} = \sum_{t=1}^{L}\sum_{j=1}^{M}\sum_{i\in\mathcal{C}_j^{(+)}}\left(1-x_i^{(t-1)}\right)y_j^{(t)}.$$

For the scheduling problems we consider here, the QUBO simplifies to

$$H = H'_{\text{no-op}} + H'_{\text{precond}} + H'_{\text{effects}} + H_{\text{conflict}}.$$

CNF-based Mapping

Besides direct mapping, we also experimented with getting the QUBO encoding by first mapping planning problems to SAT (in CNF form) and then using the known approach to map the resulting CNF encoding to QUBO.

A SAT's conjunctive normal form (CNF) expression over n Boolean variables $\{x_i\}$ consists of a set of clauses $\{C_a\}$ each consisting of k variables, possibly negated, connected by logical ORs:

$$b_1 \vee b_2 \vee \cdots \vee b_k,$$

where

$$b_i \in \{x_1, x_2, ..., x_n, \neg x_1, \neg x_2, ..., \neg x_n\},$$

and the number of variables k in the clause can vary from clause to clause. In a CNF, all of the clauses must be satisfied, which means they are connected by an AND operator.

We used the first of the four PDDL to CNF translators built into the SATPLAN planner (Kautz 2004). This "action-based" encoding starts with the time-slice encoding approach and then further removes all variables representing state variables while adding constraints that capture the relationships between actions in consecutive time steps that were previously enforced by relationships between actions and state variables. We chose this encoding because it tends to produce the smallest SAT encodings. While it may not be the easiest to solve by a SAT solver, it's more likely to be translatable to a QUBO that can fit within our very limited number of available qubits in the D-Wave machine.

We convert a CNF instance to QUBO by first transforming it to Polynomial Unconstrained Binary Optimization (PUBO), a generalization of QUBO in which the objective function is a pseudo-Boolean of arbitrary degree. For each clause in a given CNF instance, we introduce a term to the PUBO instance equal to the conjunction of the negation of all the literals in that clause. Thus, an original negative literal is replaced by the corresponding binary variable and a positive literal is replaced by the difference of one and the corresponding binary variable. For example, the CNF clause $(x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4)$ would correspond to the PUBO term $(1 - x_1)x_2x_3(1 - x_4)$.

We then reduce higher degree terms in the PUBO instance using an iterative greedy algorithm that is related to one described in (Boros and Hammer 2002). At each step, the pair of variables that appears in the most terms is replaced by an ancilla variable corresponding to their conjunction. If there are multiple such pairs, then one is chosen arbitrarily. A penalty term is introduced to enforce that the ancilla variable indeed corresponds to the requisite conjunction. For example, to reduce the degree of a term $x_1x_2x_3$, we may introduce an ancilla variable y_{12} that we will encourage to equal x_1x_2 by using a penalty term $3y_{12}+x_1x_2-2x_1y_{12}-2x_2y_{12}$, which is 0 if $y_{12} = x_1x_2$ and > 0 otherwise. The term $x_1x_2x_3$ is removed from the PUBO and replaced with $y_{12}x_3 + 3y_{12} + x_1x_2 - 2x_1y_{12} - 2x_2y_{12}$. The penalty weight we use is equal to one plus the greater of the sums of the magnitudes of the positive coefficients and negative coefficients of the terms the ancilla is used to reduce (Babbush, O'Gorman, and Aspuru-Guzik 2013). The one is added to ensure that the constraint-satisfying solutions have lower total cost than the constraint-violating solutions. One is convenient, and in keeping with the integer coefficients

[1] A less stringent way to avoid overcompensating would be to add this penalty only when the effect changes the variable, as we have done in the no-op term. The problem is that natively that is not a quadratic term. Of course one could then reduce that term, but here we choose to use the more stringent solution.

for the other terms, but any positive constant would do. This procedure is repeated until the resulting PUBO is quadratic.

Experimental Setup

To evaluate our approach, we use the benchmark set of planning instances based on graph-coloring (Rieffel et al. 2014a) that consist of parametrized families of hard planning problems. Having parametrized families enables the investigation of scaling behavior using small problems, which is crucial for evaluating early technology that is not mature enough to run real-world problems. Even when setting the encoding horizon of the graph-coloring to 1 to fit them onto the 512 qubits available in current quantum annealers, the problems can still be considered hard. Note that vertex coloring, an NP-complete problem, is strongly related to the core scheduling aspect of many planning applications (Chien et al. 2012). A scheduling problem in which no pair of tasks can be assigned the same time-slot is analogous to a coloring instance in which the the tasks are vertices, conflicts are edges, and the minimum makespan is the chromatic number, i.e. the minimum number of colors necessary to color each of the vertices such that no two adjacent ones have the same color.

Because of the overhead in mapping and embedding planning problems, even the smallest IPC problems (IPC 2004) are more than an order of magnitude too large to be run on the current D-Wave device. Therefore, we currently do not include the any result on existing IPC benchmarks.

Vertex Coloring as Planning

Given an undirected graph $G = \{V, E\}$ with n vertices and k colors, the vertex coloring problem asks for a solution in which: (1) all vertices are colored and (2) any pair of vertices connected by an edge is colored differently. The corresponding planning problem is as follows: for each vertex v there are:

- k actions a_v^c representing coloring v with color c;
- A 'goal variable' s_v^g representing whether or not v has been colored at all; and
- A state variable s_v^c representing whether or not v has been colored with the color c.

Let $C(v)$ be the set of neighboring vertices that are connected to v by an edge. For each action a_v^c, there are $|C(v)|+1$ preconditions: (1) $s_v^g = F$, which indicates that v is not already colored; and (2) for each $w \in C(v)$, $s_w^c = F$, guaranteeing that none of neighboring v_i are already colored with color c.

Each action a_v^c has two effects: $s_v^g = T$ and $s_v^c = T$.

In the initial state, none of the vertices are colored: $\forall v \in V, \forall c \in [k] : s_v^g = F$, and $s_v^c = F$. The goal state requires that all vertices are colored: $\forall v \in V : s_v^g = T$. A plan is a sequence of n actions, each of which colors a vertex v.

Problem generation: We parametrically generate instances using Erdös-Rényi model of random graphs $G_{n,p}$, where n is the number of vertices and p is the probability of an edge between each pair of vertices, using an extension of Culberson et al.'s (Culberson, Beacham, and Papp 1995) graph generator program. Our extension generates PDDL files (Rieffel et al. 2014b), at the phase transition $c = pn = 4.5$ (Achlioptas and Friedgut 1999; Dubois and Mandler 2002; Achlioptas and Moore 2003; Coja-Oghlan 2013).

We preset the number of colors to $k = 3$ and for that k value the maximum problem size that we can embed in the D-Wave Two machine with 509 qubits is $n = 16$. Specifically, for each of $n = 8, 9, \ldots, 16$, we use 100 solvable problem instances at the phase transition for each size. For $n = 12$ to $n = 16$, we reuse problems generated in (Rieffel et al. 2014a), and for $n = 8$ to $n = 11$, we generate new ones using the same approach.

For each generated instance, we then generate 3 different QUBOs, each described in the previous sections: (i) direct mapping; (ii) time-slice mapping; and (iii) CNF-based mapping.

Embedding: From a mapped QUBO instance, we generate a vertex model by running D-Wave's heuristic embedding software (Cai, Macready, and Roy 2014) on the mapped QUBO instance, using the software's default parameters. The output of the embedding software is a set of pairwise-disjoint, connected vertex models $\{C_i\}$ in the hardware graph corresponding to the variables $\{z_i\}$ in the original QUBO, which will then be converted to Ising form to be run on the D-Wave machine. Before running, the Ising is rescaled so that all coefficients are between $[-1, 1]$. We performed our own parameter setting following (Rieffel et al. 2014b), rather than using D-Wave's defaults. The parameter settings for these runs are discussed in detail in (Rieffel et al. 2014b).

Solving: All quantum annealing runs were performed on the 509-qubit D-Wave Two machine housed at NASA Ames. While debate continues as to the best physical model for D-Wave machines (Johnson et al. 2011b; Boixo et al. 2013; Smolin and Smith 2013; Wang et al. 2013; Boixo et al. 2014; Shin et al. 2014b; Vinci et al. 2014; Shin et al. 2014a), these machines provide the first opportunity for researchers to experiment with quantum annealing. In all cases, we used an annealing time of 20 μsec, the shortest available on the hardware, though evidence suggests that a shorter time may be optimal for problems of the present size. For each embedded QUBO instance, we performed $45,000$ anneals using each of ten gauges (i.e. local symmetry transformations that leave the objective function invariant but physically change the effect of biases (Perdomo-Ortiz et al. 2015)), for a total of $450,000$ anneals per QUBO instance.

Because all of the problems we consider are solvable, we know the ground state energy (i.e., optimal value for the function q in Equation 1) in all cases; zero, the minimal value of the QUBO in all cases is attainable, and from that we can compute the ground state energy of the embedded Ising problem that was actually run. (Even for unsolvable instances, the quantum annealer would return solutions in exactly the same way it does when it fails to find a schedule when it exists.) For each embedded instance, once we obtain

the $450,000$ results from the run, we check how many times the ground state energy was obtained, which gives us the probability of solution r for a 20 μsec anneal. We then compute the expected number of runs $R = \frac{\ln(1-0.99)}{\ln(1-r)}$ required to obtain a 99% success probability, multiply by the anneal time of 20 μsec, and report $20 \times R$ μsec, the expected total anneal time to obtain a 99% success probability. We are effectively using a 0.9 sec. cutoff time, since the expected anneal time when only one anneal solves is 0.9 secs. Given that classical planners solve these problems in less than 0.1 secs., with the best planners for these problems solving them in less than 0.01 secs. (Rieffel et al. 2014a), this cutoff time seems reasonable.

We report the median expected total anneal time across 100 instances, with error bars corresponding to the 35th and 65th percentiles. Thus each data point shown represents 45 million anneals. While the total annealing time for each point is only 90 seconds, because the process to read-out the state of all qubits (i.e. solution extraction) takes considerably longer than the anneal time, and because of shared use of the machine, the wall clock time to obtain a single data point is hours not minutes. Finding the embedding, by far the longest step in the process, can take minutes for the largest instances, but fortunately needs to be performed only once per QUBO instance.

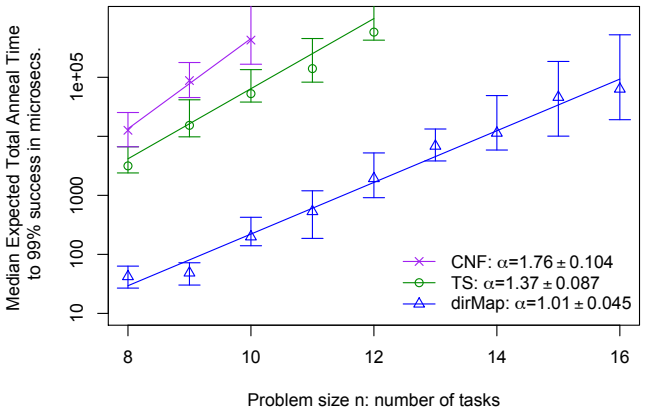

Figure 3: **Comparison of the median expected total anneal time to 99% percent success for the two mappings.** Each data point shows the median expected total annealing time to achieve 99% success over the 100 problems of each size given on the x-axis. The error bars are at the 35th and 65th percentiles. When at least half of the instances do not solve within the 0.9 sec. effective cutoff time, we no longer show the point. Also, when fewer than 65% solve, the top of the error bar is indeterminate, as happened for the last point shown in both the CNF and time-slice series.

Results and Analysis

Fig. 3 shows the relative performance, in terms of median expected total annealing time for 99% percent success, of the D-Wave Two on the family of graph coloring-based planning problems described above. When at least half of the instances do not solve within the 0.9 sec. effective cutoff time, we no longer show the point. For the CNF mapping, that happens by problem size 11. For the time-slice instances, at least half do not solve within the cutoff time by problem size 13. The figure also shows the performance using a direct map of graph coloring to QUBO.This direct mapping performs better than both of the general mappings for planning problems; it is more compact (due to its being specific to this type of problem) and likely benefits from an homogeneous parameter setting (Venturelli et al. 2014), as it generates a more uniform distribution of vertex model sizes (see Fig. 5) than the other two mappings.

There is a substantial difference between the performance on the time-slice instances and the CNF instances, with the median expected total annealing time to achieve 99% success being about a factor of 5 greater for the CNF instances than the time-slice instances (Fig. 3). The scaling for the time-slice approach is also significantly better than for the CNF approach, with an α value of 1.37 rather than 1.76 (though the scaling is estimated on very few data points).

QUBO size: (Rieffel et al. 2014b) compared some straightforward properties of both the mapped and embedded QUBOs for the two mappings, but these simple properties were all sufficiently similar across the two mappings that they could not account for so marked a difference in performance. In summary, the time-slice and CNF mappings yield comparably-sized QUBOs, with similar numbers of couplings. For problem size n, the time-slice mapping yields a QUBO of size $8n$ qubits. The CNF approach yields variable size mapped QUBOs, with the median size CNF QUBO over 100 problems only 4–8 qubits larger than the median size of the time-slice QUBOs for problem sizes 8–12. This slight difference in size cannot account for the difference in performance because for larger size problems, when the time-slice QUBOs begin to exceed the CNF QUBOs in size, the performance of the former is still better. Similarly, the median number of couplings for the CNF QUBOs exceeds that of the time-slice QUBOs by only 8–16 for problem sizes 8–12.

The most obvious properties of the embedded QUBOs are also similar. The median embedding sizes of the CNF QUBOs are only 7–28 qubits larger than the embedded time-slice QUBOs in this range, no more than a 10% difference. Large embedded vertex models contribute to poor performance, but the median (over the 100 problems) average vertex model size, and the median 90th percentile vertex model size of the embedded QUBOs for the two different mappings are virtually indistinguishable. A small difference between the median maximum vertex model sizes is seen, but it is not statistically significant. Furthermore, throughout the size range tested, the median median vertex model size – the median over the 100 problem instances of the median vertex model size of each instance – and even the median 65th percentile vertex model size, for both mappings is 1.

Deeper analysis: We took a deeper look at the distributions

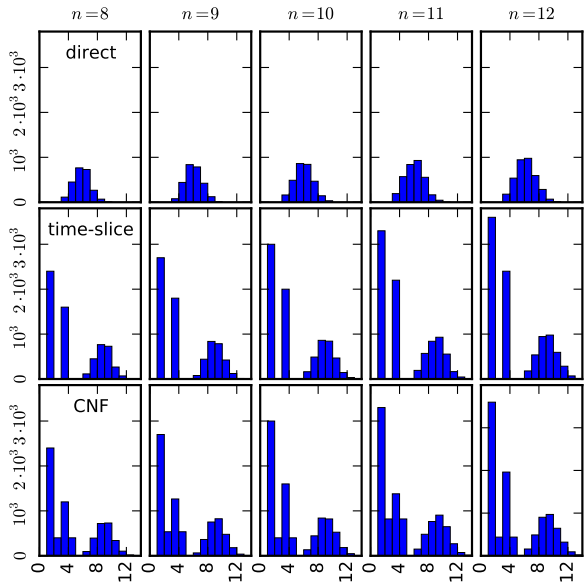

Figure 4: **Vertex degree histogram.** Histograms of the vertex degrees for the mapped QUBO graphs for each problem size under the three mappings: direct mapping, time-slice mapping, and CNF mapping.

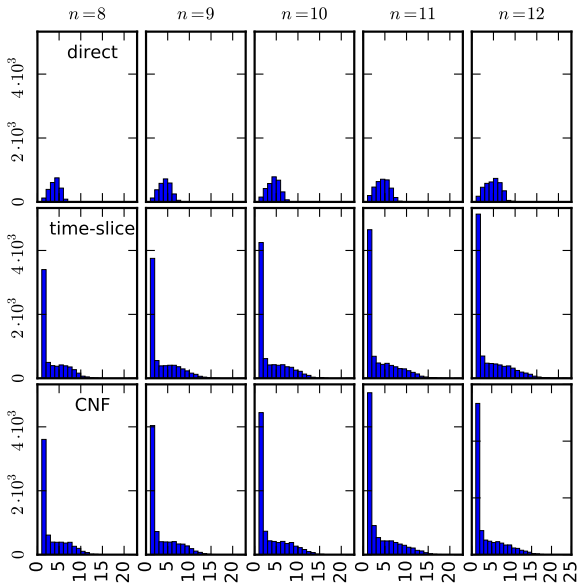

Figure 5: **Vertex model size histogram.** Histograms of the vertex model sizes in the embedded QUBOs for each problem size under the three mappings: direct mapping, time-slice mapping, and CNF mapping.

of various simple properties related to the mapped and embedding QUBOs arising from the two mappings. We also show the distributions for the direct map for comparison. In the mapped QUBOs, we looked at the distribution of of vertex degrees (the number of quadratic terms in which a variable z_i appears in the mapped QUBO). As can be seen in Fig. 4, the histograms for the time-slice and CNF mappings are very similar, while they differ markedly from the histograms for the direct map. In the embedded QUBOs, we looked at the distribution of the vertex model sizes (Fig. 5) and also the distribution of the graph diameter of the vertex models (not shown), but found little difference between the distributions for the time-slice and CNF embedded QUBOs. Therefore these properties can contribute at most a small amount to the performance difference between the two mappings.

We begin to see differences when we look at distributions of the coefficients in the mapped QUBOs. Fig. 6 shows histograms of the coefficient of the mapped QUBOs (Equation 1) , converted to Ising, and rescaled so that all of the h_i and J_{ij} coefficients are between $[-1, 1]$. Let $j_i = \sum_j J_{ij}$. Fig. 7 shows a histogram of the h_i and j_i. Both histograms show significant differences between the two mappings.

Another potential origin of the performance difference is the topology of the vertex models. A quick analysis showed that for all three mappings nearly all ($> 99\%$) of the vertex models of the embedded QUBOs are trees. We intend to do a further classification of the graph structures, and to examine differences in the frequency of different structures between the mappings.

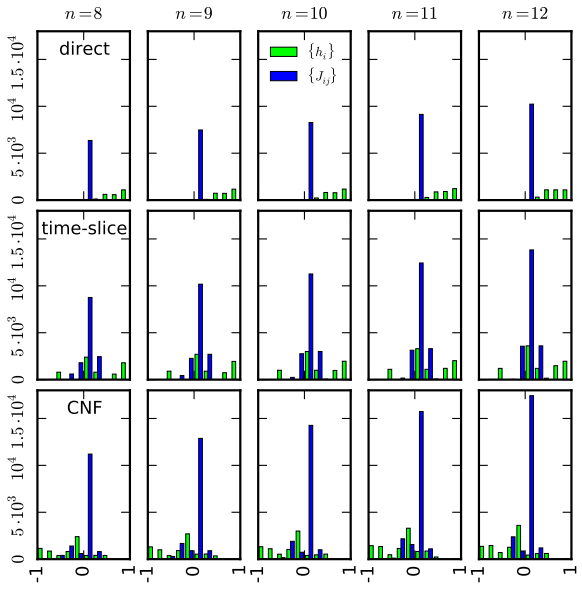

Figure 6: **QUBO coefficient histogram.** Histograms of the h_i and J_{ij} in the mapped QUBOs for each problem size under the three mappings: direct mapping, time-slice mapping, and CNF mapping.

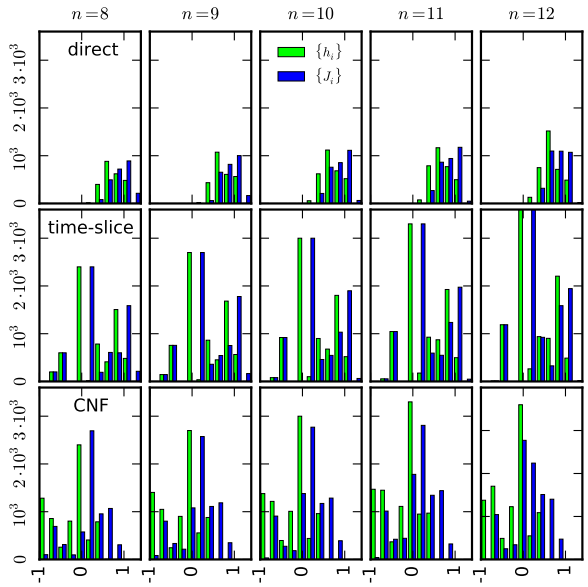

Figure 7: **Histogram of h_i and $j_i = \sum_j J_{ij}$.** Histograms of the h_i and $j_i = \sum_j J_{ij}$ in the mapped QUBOs for each problem size under the three mappings: direct mapping, time-slice mapping, and CNF mapping.

Conclusions and Future Work

We show how quantum annealing can solve planning problems via mapping to QUBO. We introduced two general mapping techniques and applied them to planning problems based on graph-coloring. We ran these problems on an early quantum annealer and saw significant performance differences. We began an investigation of various properties of the mapped and embedded QUBOs to understand which properties do and do not contribute to the performance differences.

In the future, we will examine the differences in distribution of coefficients and the topology of vertex-models to generate hypotheses regarding properties that could explain the performance differences. To test these hypotheses, we will generate small, artificial instances capturing those properties and evaluate the annealer's performance. For the properties that pass this initial test, we will perform a statistical analysis of the correlation between them and the performance of the annealer on the family of scheduling-type problems.

While at this early stage there is no advantage in solving STRIPS planning problems via quantum annealing over classical compilation approaches such as SAT, CSP, or MILP, we believe quantum annealing, and especially compilation techniques for quantum annealing, are both worth exploring even on primitive quantum hardware for several reasons. First, certain quantum algorithms have been proven to outperform classical algorithms on classes of problems of practical interest, sometimes, as for factoring, reducing the complexity from superpolynomial to polynomial. Many of the most useful classical algorithms in use today are heuristic algorithms, which have not been mathematically proven to outperform other approaches, but have been shown to be more effective empirically. Until recently, it was not possible to explore existing quantum heuristic algorithms, because without quantum hardware an empirical analysis could not be done. One of the biggest open questions in quantum computing is the breadth of its applications, with the potential of heuristic quantum algorithms, such as quantum annealing, being the biggest unknown. Major hardware development efforts are underway to build better quantum computational hardware. In order to fully explore the potential of this hardware, we must understand how best to compile practical problems to a form that is suitable for quantum hardware.

While early quantum annealing hardware can handle only small instances, by analyzing the results obtained under these limitations, we can nevertheless gain insights into the best programming and compilation techniques for quantum annealers, and ultimately into the potential of quantum annealing to solve problems of practical interest in planning and scheduling and beyond.

Acknowledgements

The authors are grateful to Zhihui Wang for helpful discussions and feedback on the draft, and to Vadim Smelyanskiy for useful discussions and support. This work was supported in part by the Office of the Director of National Intelligence (ODNI), the Intelligence Advanced Research Projects Activity (IARPA), via IAA 145483; by the AFRL Information Directorate under grant F4HBKC4162G001. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ODNI, IARPA, AFRL, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purpose notwithstanding any copyright annotation thereon. The authors also would like to acknowledge support from the NASA Advanced Exploration Systems program and NASA Ames Research Center and NASA grant NNX12AK33A.

References

Achlioptas, D., and Friedgut, E. 1999. A sharp threshold for k-colorability. *Random Structures and Algorithms* 14(1):63–70.

Achlioptas, D., and Moore, C. 2003. Almost all graphs with average degree 4 are 3-colorable. *Journal of Computer and System Sciences* 67(2):441–471.

Babbush, R.; O'Gorman, B.; and Aspuru-Guzik, A. 2013. Resource efficient gadgets for compiling adiabatic quantum optimization problems. *Annalen der Physik* 525(10-11):877–888.

Blum, A., and Furst, M. 1997. Planning through planning graph analysis. *Artificial Intelligence Journal* 90:281–330.

Boixo, S.; Albash, T.; Spedalieri, F. M.; Chancellor, N.; and Lidar, D. A. 2013. Experimental signature of programmable quantum annealing. *Nature communications* 4.

Boixo, S.; Rønnow, T. F.; Isakov, S. V.; Wang, Z.; Wecker, D.; Lidar, D. A.; Martinis, J. M.; and Troyer, M. 2014. Evidence for quantum annealing with more than one hundred qubits. *Nature Physics* 10(3):218–224.

Boros, E., and Hammer, P. L. 2002. Pseudo-boolean optimization. *Discrete applied mathematics* 123(1):155–225.

Cai, J.; Macready, B.; and Roy, A. 2014. A practical heuristic for finding graph minors. arXiv:1406:2741.

Chien, S.; Johnston, M.; Frank, J.; Giuliano, M.; Kavelaars, A.; Lenzen, C.; Policella, N.; and Verfailie, G. 2012. A generalized timeline representation, services, and interface for automating space mission operations. In *12th International Conference on Space Operations*.

Choi, V. 2008. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. *Quantum Information Processing* 7(5):193–209.

Coja-Oghlan, A. 2013. Upper-bounding the k-colorability threshold by counting covers. arXiv:1305.0177.

Culberson, J.; Beacham, A.; and Papp, D. 1995. Hiding our colors. In *Proceedings of the CP95 Workshop on Studying and Solving Really Hard Problems*, 31–42.

Das, A., and Chakrabarti, B. K. 2008. Colloquium: Quantum annealing and analog quantum computation. *Rev. Mod. Phys.* 80:1061–1081.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into csp. *Artificial Intelligence Journal* 132(2):151–182.

Dubois, O., and Mandler, J. 2002. On the non-3-colourability of random graphs. arXiv:math/0209087.

Farhi, E.; Goldstone, J.; Gutmann, S.; and Sipser, M. 2000. Quantum computation by adiabatic evolution. arXiv:quant-ph/0001106.

Fikes, R. E., and Nilsson, N. J. 1972. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3):189–208.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning: theory & practice*. Elsevier.

2004. The international planning competition website. http://icaps-conference.org/index.php/Main/Competitions.

Johnson, M. W.; Amin, M. H. S.; Gildert, S.; and et al. 2011a. Quantum annealing with manufactured spins. *Nature* 473:194–198.

Johnson, M.; Amin, M.; Gildert, S.; Lanting, T.; Hamze, F.; Dickson, N.; Harris, R.; Berkley, A.; Johansson, J.; Bunyk, P.; et al. 2011b. Quantum annealing with manufactured spins. *Nature* 473(7346):194–198.

Kautz, H. A., and Selman, B. 1999. Unifying sat-based and graph-based planning. In *Proceedings of IJCAI'1999*.

Kautz, H. 2004. Satplan04: Planning as satisfiability. *Working Notes on the Fourth International Planning Competition (IPC-2004)* 44–45.

Lucas, A. 2013. Ising formulations of many NP problems. arXiv:1302.5843.

Nielsen, M., and Chuang, I. L. 2001. *Quantum Computing and Quantum Information*. Cambridge: Cambridge University Press.

Perdomo-Ortiz, A.; Fluegemann, J.; Biswas, R.; and Smelyanskiy, V. N. 2015. A performance estimator for quantum annealers: Gauge selection and parameter setting. *arXiv preprint arXiv:1503.01083*.

Rieffel, E. G., and Polak, W. 2011. *A Gentle Introduction to Quantum Computing*. Cambridge, MA: MIT Press.

Rieffel, E. G.; Venturelli, D.; Hen, I.; Do, M.; and Frank, J. 2014a. Parametrized families of hard planning problems from phase transitions. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-14)*, 2337 – 2343.

Rieffel, E. G.; Venturelli, D.; O'Gorman, B.; Do, M.; Prystay, E.; and Smelyanskiy, V. N. 2014b. A case study in programming a quantum annealer for hard operational planning problems. To Appear in Quantum Information Processing, and arXiv:1407.2887.

Shin, S. W.; Smith, G.; Smolin, J. A.; and Vazirani, U. 2014a. Comment on "Distinguishing classical and quantum models for the D-Wave device". arXiv:1404.6499.

Shin, S. W.; Smith, G.; Smolin, J. A.; and Vazirani, U. 2014b. How "quantum" is the D-Wave machine? arXiv:1401.7087.

Smelyanskiy, V. N.; Rieffel, E. G.; Knysh, S. I.; Williams, C. P.; Johnson, M. W.; Thom, M. C.; Macready, W. G.; and Pudenz, K. L. 2012. A near-term quantum computing approach for hard computational problems in space exploration. arXiv:1204.2821.

Smolin, J. A., and Smith, G. 2013. Classical signature of quantum annealing. arXiv:1305.4904.

Venturelli, D.; Mandrà, S.; Knysh, S.; O'Gorman, B.; Biswas, R.; and Smelyanskiy, V. 2014. Quantum optimization of fully-connected spin glasses. *arXiv preprint arXiv:1406.7553*.

Vinci, W.; Albash, T.; Mishra, A.; Warburton, P. A.; and Lidar, D. A. 2014. Distinguishing classical and quantum models for the D-Wave device. arXiv:1403.4228.

Wang, L.; Rønnow, T. F.; Boixo, S.; Isakov, S. V.; Wang, Z.; Wecker, D.; Lidar, D. A.; Martinis, J. M.; and Troyer, M. 2013. Comment on "Classical signature of quantum annealing".